Mandarin Tone Trainer

Mandarin Tone Trainer

Daniel Gaspari

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2016

Abstract

Because Mandarin Chinese uses semantic tones to convey meaning, it can be challenging for new students of the language. Learning to speak these tones correctly requires practice and adequate feedback. Today's online learning platforms offer instruction in Mandarin, but they lack a good way for students to practice speaking tones.

I have built an interactive web application to help students practice these tones using only a browser and a microphone. It shows the pitch of the student's voice while speaking various practice words. The application also presents a native speaker's pitch graph for comparison. The Mandarin Tone Trainer offers an interactive way to learn Mandarin, and because it works within a browser it can be easily integrated with existing online courseware.

Dedication

This project truly began when my friend, Jonathan Hsu, moved to Taiwan. I'd like to dedicate this thesis to him. Hopefully I've learned enough Mandarin that I'll be able to converse with his family the next time I visit.

Acknowledgments

I'd like to thank everyone who volunteered to record spoken samples of Mandarin for my project, especially Marilyn Bao and the students in her Mandarin class. My advisor, Paul Bamberg, was a great help during this project. I started knowing nothing about signal processing, and with his help I was able to jump right into this project. I'd also like to thank Dr. Jeff Parker for his patient guidance.

Table of Contents

List of Figures

Chapter 1 Introduction

The idea of learning foreign languages online is appealing. Massively Open Online Courses, or MOOCs, can teach new languages to online students anywhere in the world. But these online courses are hampered by a lack of corrective feedback, which is especially important to learning a new language like Mandarin Chinese (Ventura, 2014). Chinese languages use lexical tones to convey different meanings for the same utterance. Learning how to properly speak with these tones is a critical part of the language, but it can be difficult when learning online. Typically an online learner can only practice in isolation without gaining feedback on correct usage of tones. My web application, Mandarin Tone Trainer, allows students to practice tones with immediate feedback. Such a tool could make online courses in Mandarin more useful to students.

1.1 Mandarin Tones

Languages with lexical tones have varying meanings for the same spoken sounds. In the Standard Mandarin dialect of Chinese, a syllable may have one of four different tones applied to it, or it may be pronounced tonelessly. Each tone is distinguished by the pitch contours involved. As seen in Figure 1, the same syllable can mean four different words depending on the tone that is used.

Figure 1. The register and contours of the four major tones of Mandarin Chinese.

For students learning Mandarin as a second language, learning how to speak tones correctly is absolutely necessary in conveying the meaning of each word. But for students whose primary language does not use tones in this way, this can be a difficult transition. English speakers, for example, may use tones to convey grammatical context (e.g. raising the voice when asking a question), but the words themselves maintain the same meaning. Therefore, students learning Mandarin who are unfamiliar with lexical tones must practice quite a bit in order to pick up on these subtle changes. If students are taking courses in person, the instructor can guide students as they practice speaking. Online students do not have such feedback and need alternate methods to learn tones.

1.2 Massively Open Online Courses (MOOCs)

Massively open online courses are a popular type of online learning platform. They are academic courses offered entirely online to students with a wide range of skill levels. Many institutions, such as Harvard University and MIT, are establishing platforms

to offer MOOCs on many subjects, but they have their limitations (Chang, Hung, & Lin, 2015).

Due to the open nature of MOOCs, the students involved in a course may freely come and go. Participating students may also possess prior skill in the given topic. While it might be possible for fellow students to practice speaking to each other online, it would be very difficult to group students in a massive and open course. Students recognize the importance of interaction and practice when learning a new language, but it is challenging to offer these opportunities to students participating in a MOOC. (Bárcena, Martín-Monje, & Castrillo, 2014).

1.3 Objectives

Someone who wants to learn Mandarin online will find it difficult to learn proper pronunciation without some kind of feedback. Dorothy Chun at the University of Texas has shown in multiple studies that having visual feedback while learning Mandarin tones can be beneficial to understanding the language. Her studies had groups of students learn tones while being provided with visual feedback of their vocal pitch. Chun's studies showed that these groups performed better than control groups at learning tones; therefore MOOCs could benefit from having interactive tools like those used by Chun (Chun, 1989, 2013).

My project offers a solution which could be integrated with online courses in Mandarin. It is a standalone web application that lets users practice by speaking example words, then receiving immediate visual feedback. The application first shows a graph of a native Mandarin speaker's pitch when speaking an example. The student is then

prompted to speak the word into a microphone. After the student speaks and the audio sample has been processed, the application overlays the user's vocal pitch onto the native speaker's pitch graph. The visual feedback will help students learn how to properly speak Mandarin tones. A user needs a microphone and a WebRTC-enabled browser for the application to function, but no additional software is necessary, making the application easy to assimilate into online courses.

1.4 Overview

This document provides details on the process and results of my project. Chapter 2 provides an overview of online courses in Mandarin that exist today and the types of software available for practicing tones. Chapter 3 discusses the prerequisites for my project in terms of developing a pitch tracker and building an application meant for online classrooms. Chapter 4 describes the Mandarin Tone Trainer application in detail, and the final chapter concludes with the results of my project.

Chapter 2 Prior Work

When I began this project I wanted to see what kind of interactivity was available in today's online classrooms. I also wanted to see what sort of pitch tracking software was available for students studying Mandarin. In the following sections I describe my findings and the justification for going forward with my Mandarin Tone Trainer project.

2.1 MOOCs

There are a few online courses that currently offer instruction in Mandarin. Each of these belongs to an online platform for massively open courses.

2.1.1 Coursera: Chinese for Beginners

Coursera is a MOOC platform run by a for-profit company which provides certifications for various fields of study. They do offer some courses for free, such as their introductory Mandarin course. The course is broken into a number of ten minute video lessons, accompanied by multiple choice quizzes.

These quizzes are the only interactive portion of the course. Students watching the lessons can repeat along with the instructor in the video, but there is no way to gain feedback on proper pronunciation.

2.1.2 Udemy: Essential Chinese for Travelers

Udemy is a MOOC platform where online courses can be submitted by individual users. These courses might be free or require payment. They typically involve a single instructor that has created video and test content on the Udemy platform. The Essential Chinese for Travelers course is one such option, providing a series of video lessons recorded by an instructor.

Figure 2. Video lesson from Udemy's Essential Chinese for Travelers course.

The Essential Chinese for Travelers course features a section about learning how to speak using tones. In the video, the instructor invites users to contact her through the

voice call application WeChat. This is an interesting way to engage online students, but it cannot scale to a massive number of people. The number of enrollees on the Udemy courses could be small enough that the instructor is not inundated with requests to practice speaking Mandarin, but this means that the course is not a true MOOC. My project's goal of offering students immediate feedback in the browser would scale better and be available to students at any time.

2.1.3 edX: Basic Mandarin Chinese - Level 1

The edX platform is a series of MOOCs offered by a number of institutions such as MIT and Harvard University. There is a free course in Mandarin available on edX. The lesson content is similar to the previously described MOOC platforms, offering readings, lecture videos, and quizzes. The course is active for a one year period. As long as the class is active, the content can be consumed at the student's own pace. The platform will provide certification to students that complete the course within the given timeframe.

The edX Mandarin MOOC does offer one interactive way to assess students. It asks that students post a recording of themselves speaking a script for each lesson. It then asks that a student perform assessment on other peers' recordings. This lets students practice speaking and get feedback, although the response will not be immediate. As can be seen in Figure 3, the peer assessment due date is months after the quiz answers are due.

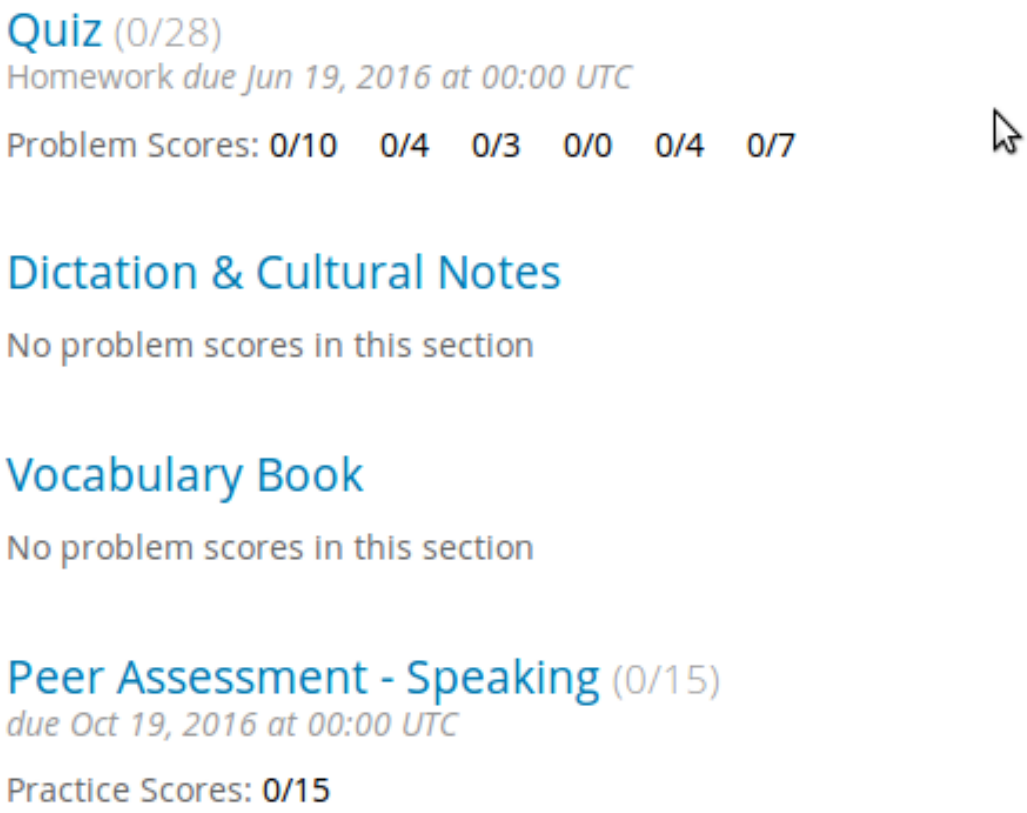

Figure 3. Details for one lesson of the edX Mandarin course.

The assignments have a generous time limit, but to complete these assignments one needs to rely on peers to assess spoken examples. Most students participating in this MOOC will not have a great deal of experience with Mandarin. Feedback from fellow students is probably not very reliable since most people taking the course will be new to the language. The designers of this MOOC have found a way to provide feedback on pronunciation that is feasible for a massive class, but a tool that students could use online at any time with examples from native speakers would be preferable.

2.2 Pitch Tracking Software

Although there do not appear to be any online pitch trackers for learning Mandarin, there are a number of software packages available for download. These software packages vary from commercial language learning products to freely available tools developed by educational institutions.

2.2.1 Pinyin Practice

Pinyin Practice is one of many online flash games available to the public for free. It is a simple way to practice identifying tones. A user is prompted with a word in Mandarin as well as an audio clip of a native speaker saying the word. The user gets to decide which of the four tones the word contains. The user does not get an opportunity to practice speaking, however. This is simply a good way to practice identifying tones spoken by native speakers.

Figure 4. An image of the online Flash game, Pinyin Practice.

2.2.2 Rosetta Stone Chinese

Rosetta Stone is a software company that produces applications for learning new languages. Rosetta Stone Chinese is a software package that instructs users in the Standard Mandarin dialect of Chinese. It has a variety of vocabulary exercises, some of which prompt the user to speak a word via a microphone. The user interface reports whether the word was spoken correctly or not.

There are some clear differences between my project and what Rosetta Stone offers. My goal is to make an application that could be integrated with online platforms, while Rosetta offers a software package that students must purchase on their own. Furthermore, there is no detailed feedback from Rosetta about how one spoke the tones in a word. Users only see a flag marking success or failure. The application is probably

using proprietary speech recognition for this functionality, and correctness of tone may not be necessary for success.

Figure 5. Spoken vocabulary practice interface in Rosetta Stone Chinese.

2.2.3 SpeakGoodChinese

SpeakGoodChinese is a freely downloadable application specifically designed for learning Mandarin tones. It was developed by a group funded by the SURF Foundation, a Dutch nonprofit. The application logic relies on a software package known as Praat, developed by Paul Boersma and David Weenink at the University of Amsterdam. Praat

offers a user interface, signal processing libraries, and a scripting language to make customizable interfaces. The SpeakGoodChinese application is built using Praat scripts.

The user interface presents a vocabulary word along with an example pitch curve that seems to be generated from an audio sample that plays. The audio sample features a computerized voice utilizing text-to-speech software to read the prompt. The user is encouraged to speak the word. The resulting pitch curve is overlaid on top with either a success or failure reported for the given word.

SpeakGoodChinese offers visual feedback of the user's pitch while speaking Mandarin tones, but it is a purely offline application. It is built using Praat scripts, which handle signal processing and the display of the user interface. Because these scripts contain display logic, they would need to be rewritten before the application could be offered online. Doing this work would involve learning Praat script syntax and then replacing any UI logic with a web application. This would be a large effort that would not be appropriate for this project. Even if this was done, it might be more difficult to integrate with MOOCs since it is a specialized scripting language unfamiliar to most web developers. One other issue is that the example words are spoken by a computerized voice. The pronunciation in these examples might be considered unnatural or confusing to native speakers.

Figure 6. Interface of SpeakGoodChinese presenting a spoken user's pitch graph.

2.2.4 Praat

As mentioned in the previous entry, Praat is a platform for audio processing that supports construction of user interfaces. For her 2013 study, Dorothy Chun used Praat to help teach students Mandarin tones. Her researchers were able to create pitch graphs based on samples from native speakers and study participants. In this case they did not make an interface comparable to SpeakGoodChinese. Study participants received no immediate feedback. They had to wait one week after an initial training and recording session. At the second session they received pitch graphs generated by researchers using

the Praat software library. Chun found that even this delayed feedback was useful for learning tones (Chun, 2013).

Using Praat purely on its own might be possible online, but as mentioned above it is a specialized platform that might discourage integration by others due to unfamiliarity. A server could run a Praat script to obtain pitch information for an audio sample, but because a third party implements the underlying logic, it might be difficult to maintain. One wouldn't be able to improve performance or functionality of the pitch tracking logic beyond adjusting the Praat script. The goal of this project is to provide a comprehensive solution that would integrate well with online platforms.

Chapter 3 Project Requirements

In order for me to build the application I envisaged, I needed to understand how to calculate vocal pitch and how to obtain audio samples on the web. In the following section I describe what I learned about pitch tracking algorithms and the Web technologies that can obtain audio within the browser. I also describe what the kind of application I wanted to build and the resources I'd need to achieve my goals.

3.1 Pitch Tracking

3.1.1 Pitch and Fundamental Frequency

In order to present feedback to a user, I needed to determine the pitch of the user's voice. This is done by obtaining the fundamental frequency of the audio input, a property of periodic auditory waveforms. The psychological relationship between fundamental frequency and pitch is well known (Gerhard, 2003). My application focuses on obtaining and presenting this information.

Fundamental frequency, which can be abbreviated as f_0, is the rate at which the largest wavelength of a periodic signal repeats. Determining the fundamental frequency involves finding the right pattern within a signal. As seen in Figure 7, there may be multiple repeating wavelengths in a periodic waveform, but the longest of these is used to determine f_0. To calculate the fundamental frequency, we take the inverse of the amount of time that passes for one cycle of this pattern:

$$f_0 = 1 / s$$

This formula shows the relationship of fundamental frequency to time. “s” represents the amount of time, in seconds, that passes for a single period of the repeating pattern.

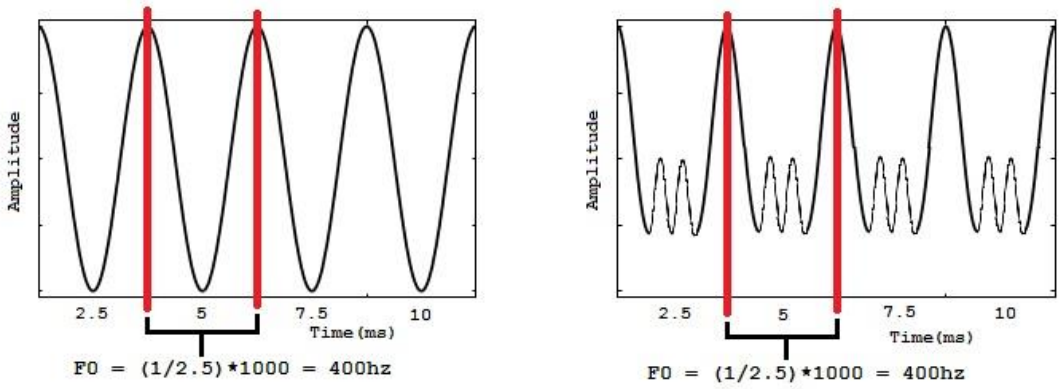

Figure 7. Two examples of how f0 is calculated for a waveform.

My application needs to calculate the fundamental frequency of human speech. There are mechanical properties of speech that affect how apparent pitch is determined. For example, human speech typically consists of voiced and unvoiced portions.

Unvoiced portions of speech involve noises that do not carry pitch, such as a “p” or “s” sound. My application would need to ignore such information when determining the pitch of a speaker’s voice. I would only want to consider voiced portions to determine the actual pitch of speech.

The vocal folds of the glottis, which are commonly known as vocal cords, only vibrate during voiced speech. The resulting vibration has a repetitive pattern called a glottal pulse. It is caused by air being expelled from the glottis. By identifying the glottal

pulses, one can determine f_0 for voiced parts of speech, and thereby determine the apparent pitch of a speaker's voice (Talkin, 1995).

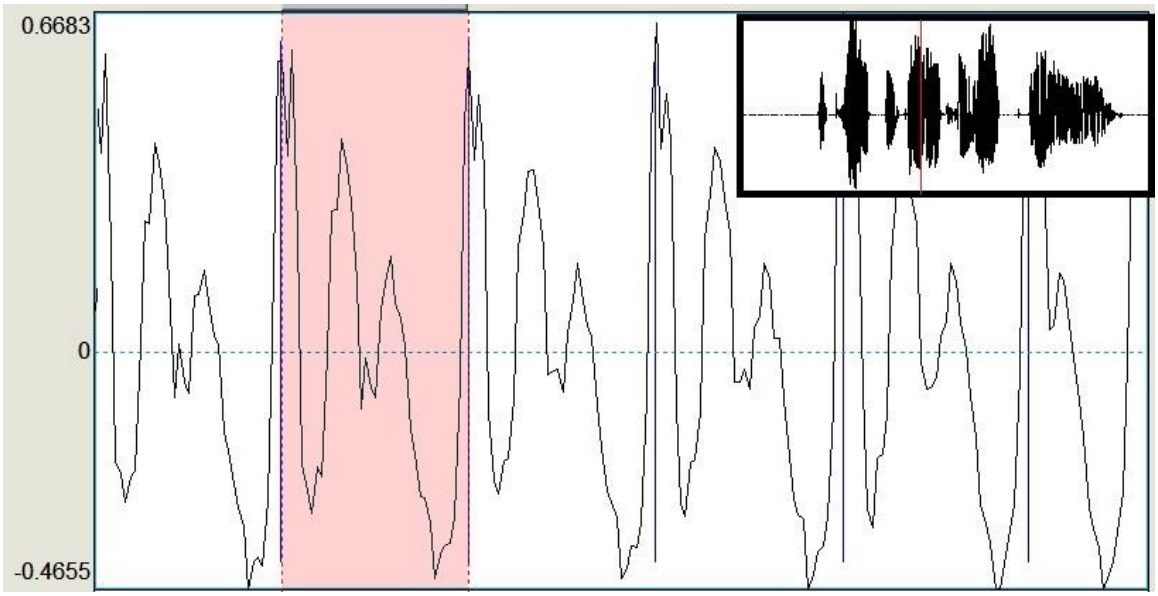

Figure 8. A highlighted glottal pulse in an audio sample of human speech.

3.1.2 Robust Algorithm for Pitch Tracking (RAPT)

David Talkin's Robust Algorithm for Pitch Tracking (Talkin, 1995), or RAPT, offers a way to estimate the fundamental frequency of an audio signal containing human speech. The algorithm has two major steps. The first step is to find potential glottal pulse patterns in an audio input. The second is to estimate whether the speaker is using voiced speech or unvoiced speech.

The algorithm's first task is to determine which periodic patterns in the audio can be used to determine the fundamental frequency. It does so by using a normalized cross correlation function, or NCCF. Cross-correlation of signals is used to search for a specific

pattern in one waveform that matches another. In this case the algorithm is looking for repeating patterns in the audio input. It will essentially scan each portion of a signal and check for repeating segments.

Once the algorithm has its candidates for potential glottal pulses, it must determine which portions of speech are actually voiced. RAPT has a dynamic programming step to determine the voicing state for each frame of the audio sample. The algorithm recursively works through each frame and determines the likelihood of whether the given speech is in a voiced or unvoiced state. If it is deemed likely that a frame is during voiced speech, the best candidate for a glottal pulse can be selected. The duration of this pulse is what we use to determine the fundamental frequency at that time.

The output of the algorithm is a best estimate of the fundamental frequency for each portion of a speech sample. This can be graphed as a line showing the approximate pitch frequency over time, as seen in Figure 9. For my project I used RAPT to obtain a graph of a speaker's pitch and then present that in a web application.

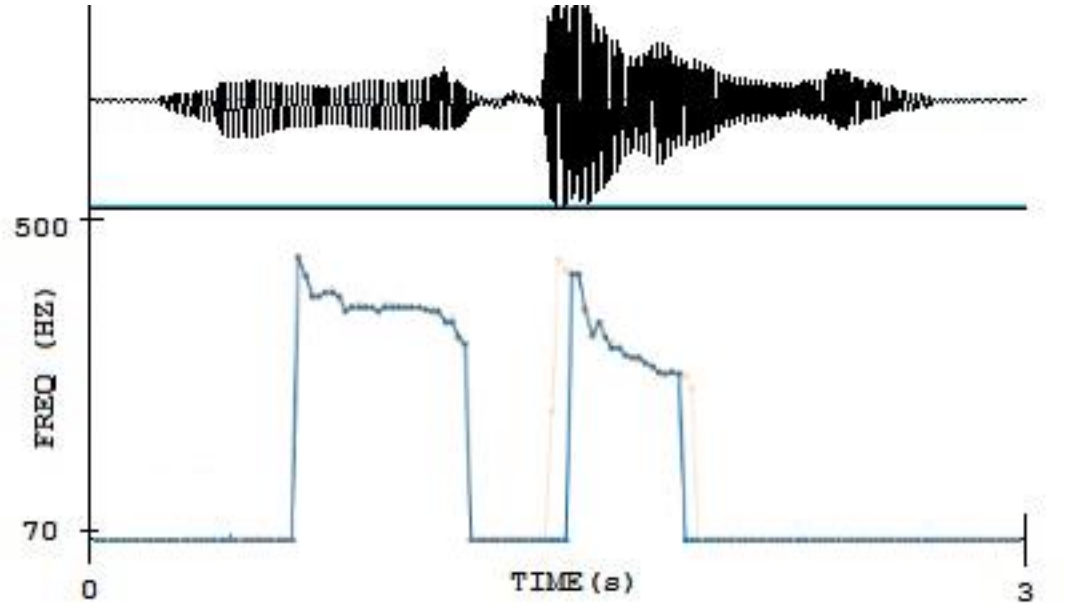

Figure 9. A potential pitch graph that plots frequency over time for a spoken audio sample.

3.2 Technology Choices

I started this project after finding a lack of interactive learning tools on MOOCs teaching Mandarin. I conceived of the Mandarin Tone Trainer as something that would benefit online students, so I decided to design a web application that could be accessed easily with no additional software for students to download. More importantly, I wanted the application to be able to integrate with existing online courseware.

Coursera, edX, and other MOOC platforms are Single Page Applications that utilize JavaScript frameworks running in the browser to handle navigation. Single Page Applications, or SPAs, are becoming a popular choice for dynamic web applications. In order to make my project integrate well with educational web content, I chose to make it a SPA as well. This meant my project needed to utilize a JavaScript framework to handle navigation and presentation of content.

I also wanted my application to be capable of running on a server without requiring a great amount of system resources. My project is hosted on virtual machines offered by Amazon Web Services. If my application could run efficiently on a small virtual server with limited processing power offered freely by Amazon, it would be suitable to run alongside online classroom applications like Coursera and edX.

3.3 Audio Input on the Web: WebRTC

For this project I needed to present users with a web interface that could record students speaking words in Mandarin and then present a pitch graph. In the past there were few options for capturing audio from users on the web. There were proprietary plugins, such as Adobe Flash, that could interact with audio input devices, but browsers did not natively support any such interaction. (Loreto & Romano, 2012). But in recent times a new web standard has been adopted by most major browsers.

WebRTC, which stands for Web Real Time Communication, is a media and communication API proposed by the W3C. The goal of the standard is to allow for real time teleconferencing between two users using a web page in a browser. The standard encompasses a variety of JavaScript APIs that a web application needs to establish a connection and stream media to another user. The standard most pertinent to this project is the MediaStream API. Browsers that support MediaStream can communicate with recording devices on a user's computer, such as a web-cam or microphone (Bergkvist, 2012). Most major browsers have incorporated the standard, and holdouts such as Microsoft and Apple are moving towards adding support (Gouaillard, 2015).

Because it is a well-established standard for capturing audio in the web, I decided to use the MediaStream API to record audio for my project. The biggest argument for using WebRTC's standard is that it is supported natively in the browser. No additional plugins need to be downloaded. For example, a well-established company like Google uses WebRTC for its Google Hangouts web application. By using WebRTC's MediaStream API I chose a platform-independent implementation that is accessible to most users on the web.

3.4 Collecting Example Audio Samples

One of the key features of my project involved presenting example words spoken by native Mandarin speakers, so I needed volunteers to obtain the necessary recorded examples. Luckily, I had a friend in Taiwan who could help. I also took part in a Mandarin language class, and my instructor was willing to record examples for my project.

To make the Mandarin Tone Trainer a tool that could be used by many students, I wanted to have example words spoken in a variety of pitch ranges. Ideally a student would work with examples spoken by someone with a comparable pitch range. I wanted to at least have one male and one female voice to choose from.

3.5 Conclusion

This chapter describes the choices I made when starting my project. I chose to use WebRTC to capture audio input within a browser. My server would take audio input from the Web and then run the data through the RAPT algorithm. It would then present a

graph of the user's pitch. The interface would need to be a Single Page Application so that it could integrate well with online courses. Once I decided upon these core criteria, I was ready to develop the Mandarin Tone Trainer.

Chapter 4 Mandarin Tone Trainer

This chapter describes the Mandarin Tone Trainer application in detail and explains the decisions I made while working on the project. It describes the system architecture and notable details of the pitch tracker algorithm and the techniques I used for capturing audio from the browser. This chapter also presents the web interface for the project.

4.1 Project Structure

This application has two main components: the web application itself, and the pitch tracker. The pitch tracker takes audio data as input and returns the data necessary to graph vocal pitch over time. While a single application could take on both of these responsibilities, I chose to work on them as separate modules.

I decided to separate the web server and pitch tracker to ensure that the pitch tracker would run efficiently and be simple to develop. I wanted to write the pitch tracker using the Python programming language, while developing the web application entirely in JavaScript. Python has become popular amongst researchers, and some have written signal processing libraries that would help me in writing my pitch tracker. Even more importantly, there are libraries like Numpy that are well suited to processing large data sets. Numpy defines its own variant of Python's array data type that allows for programs

to perform actions quickly on large sets of information (Van Der Walt, Colbert, & Varoquaux, 2011).

I anticipated that running RAPT on an audio sample would be the most computationally expensive part of my application, but I also wanted to ensure my implementation of RAPT would respond quickly to user input. With this goal in mind, I chose to implement my pitch tracker as a Python process that would be contacted by a web server running in JavaScript.

It is possible to host a web application using Node.js, which is a JavaScript runtime environment. It is a popular platform for hosting web applications, but the developer community does not have the same level of experience in signal processing. By writing my web application in JavaScript and my pitch tracker in Python, I could take advantage of each programming environment's established libraries.

I was confident I could work on these components separately and have them integrate easily thanks to the ZeroRPC library, a communication protocol that runs on the ZeroMQ queuing framework. This framework provides a socket-like interface between two applications, which can be used by two processes running on the same server to communicate with each other. It also has the flexibility to work across servers. This potential for the web server and pitch tracker to run on separate machines was another reason why I chose to keep them separate.

Since I was concerned about the performance of the pitch tracker, I wanted to ensure the potential for scalability. If a large number of students wished to use my application, a single Python process might be inundated with requests to process audio data. By making the pitch tracker a separate component, I could potentially have multiple

servers responding to pitch tracking requests. A single web server could use ZeroRPC to communicate with these separate servers. By choosing this framework for communication between the web server and pitch tracker, I made it possible for my project to handle more traffic robustly without changing the core architecture.

4.2 System Architecture

The web server runs JavaScript using Node.js via the Express library, which offers a simple API to handle HTTP requests from browsers and serve content. Since the server for my project hosts a Single Page Application, it only needs to provide one full HTML document. When a user navigates to other pages within the application, a JavaScript framework running in the browser will rewrite the markup on the page to display new content. The web server is contacted asynchronously for data as needed.

The same server also hosts a process running in Python. This process, when called with an audio sample, returns a set of points that make up the vocal pitch graph. The web application running in the browser obtains audio input from the user, passes it to the web server, which then contacts the Python process. The Python process returns the pitch graph, which the web server receives and sends back to the client. Then the end user sees a pitch graph for the audio input appear on the screen. See Figure 10 below to see the components that comprise this system and how using ZeroRPC, the pitch tracker can run on the same web server or on a separate server.

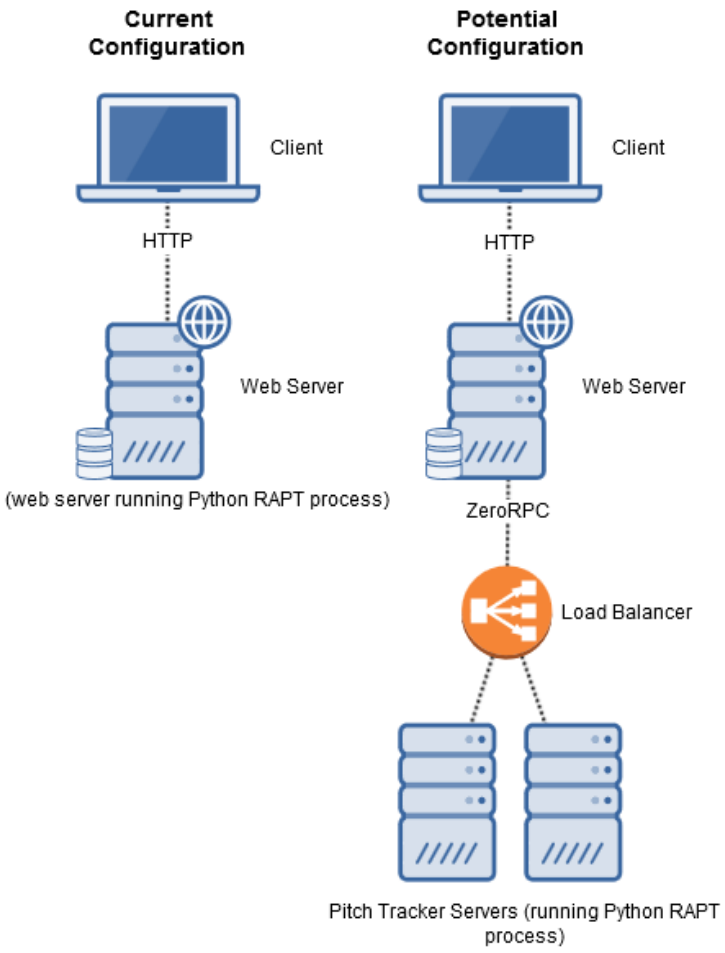

Figure 10. Current and potential system architecture of Mandarin Tone Trainer.

4.3 RAPT Implementation

I began work on my project by writing an implementation of RAPT in Python. As mentioned previously, there are two major steps to the algorithm. The first is to identify repeating patterns in the audio sample waveform which could be glottal pulses. The second step is to identify which portions of the sample involve voiced speech.

4.3.1 Normalized Cross Correlation Function

RAPT uses a normalized cross-correlation function to find potential glottal pulses. The NCCF provides a way to identify repeating patterns in a waveform. The data that makes up an audio sample can be partitioned into discrete portions of data, or frames. For each frame of the audio sample, we can see if there are any portions of data that match the same waveform in the current frame. The NCCF checks the data that follows a frame for a certain lag value range. The function will return the correlation value of each lag value at a given frame. The more similar the waveform is to the current frame, the better the correlation value. As seen below in Figure 11, the portion of the waveform highlighted in red correlates well with a portion of the sample that lags afterward. This close match will result in a higher correlation value for that lag. This high correlation implies that there is a repeating waveform at the given lag. The temporal distance of this lag between waveforms is exactly what we need to calculate the vocal pitch if this is a glottal pulse.

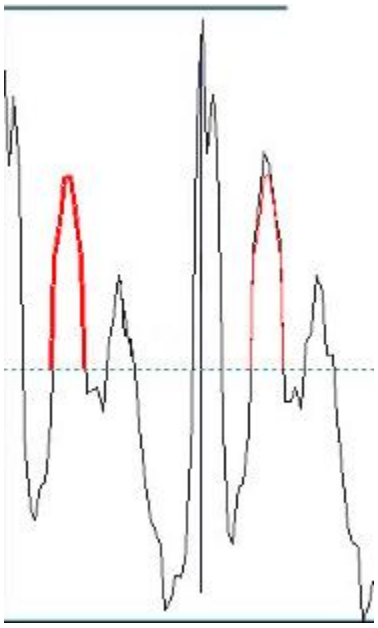

Figure 11. A highlighted sample frame and a well-correlated portion that follows it.

This portion of the algorithm iterates through each frame of the audio sample. At each frame, the lag values with the best correlation can be used to identify the frequency of potential glottal pulses. These lag values are the output of the NCCF. Figure 12 below shows the correlation values at each lag value per frame of a waveform. The notable red flat line represents a flat tone in one of my example audio samples.

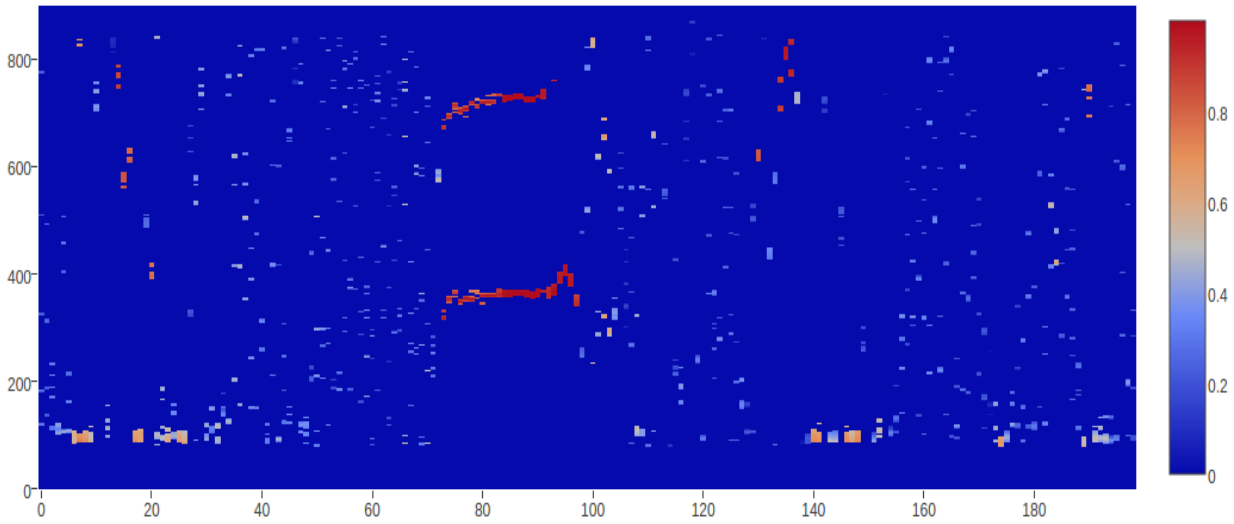

Figure 12. Correlation values returned by NCCF.

Scanning the entire audio sample for this information is computationally expensive. In David Talkin's paper on RAPT, he points out that as the sample rate of the audio input increases, the computational cost of the NCCF increases exponentially. Since it is desirable to use high quality audio when determining the vocal pitch, Talkin proposes a solution to this challenge. The first step of RAPT is to scan a lower quality version of the audio input. The original audio is downsampled to a lower sample rate. The NCCF is run on the entirety of the downsampled audio input. The next step of RAPT is to run the cross correlation function on the original audio input, but only consider lag values that had high correlation scores from the downsampled audio (Talkin, 1995). Making two passes with the cross correlation function is more computationally efficient, as I saw for myself when I was developing the algorithm in Python.

For my implementation of RAPT, I found that the performance of the algorithm depended heavily on the implementation of the NCCF. The same code is called on every

frame of the audio input, and any inefficiencies became quickly obvious. Luckily, since I was writing this algorithm in Python and storing the audio input in a Numpy array, I was able to take advantage of Numpy's sum function to handle summations. This was the resulting code for calculating the correlation values within the _get_correlation function:

```
# calculate the mean for the window, for normalizing value
# during correlation
frame_sum = numpysum(
              audio_sample[frame_start:final_correlated_sample])

mean_for_window = ((1.0 / float(samples_correlated_per_lag)) *
   frame_sum)
# select portion of the audio sample to compare
audio_slice = audio_sample[frame_start:final_correlated_sample]
# select portion of the audio at given lag to compare
lag_audio_slice = audio_sample[frame_start + lag:
                          final_correlated_sample + lag]
# convolve normalized sample from audio in frame w/ lag audio
samples = numpysum((audio_slice - mean_for_window) *
              (lag_audio_slice - mean_for_window))
# for NCCF we divide the convolved audio with the sum of each
# sample subtracted from the mean
denominator_base = numpysum((audio_slice -
   float(mean_for_window))**2)
denominator_lag = numpysum((lag_audio_slice -
   float(mean_for_window))**2)
# Only for the final 2nd pass of NCCF do we include an additional
# constant to the denominator to help deal with additional noise
# that may result from downsampling in the 1st pass
if is_firstpass:
  denominator = math.sqrt(denominator_base * denominator_lag)
else:
  denominator = ((denominator_base * denominator_lag) +
        params[0].additive_constant)
   denominator = math.sqrt(denominator)
# NCCF output for the frame - the convolved signals divided by
# the mean
return float(samples) / float(denominator)
```

Figure 13. Python code that calculates the correlation for a given lag in a sample frame.

Although I was able to make my implementation run quickly with the help of Numpy, I found that I still needed to follow the two-pass method proposed by Talkin for RAPT. When I first implemented RAPT, I tried it both with and without this two-pass method. I found that my algorithm performed much more quickly when doing two passes

over the audio input. If I used the two-pass method my RAPT implementation could return the estimated vocal pitch within a few seconds, whereas a one-pass implementation would take noticeably longer.

The only drawback to the two-pass approach was the loss of accuracy in identifying lag values with high correlation in the original audio sample. In the first pass, I collect the best lag values for a given frame of the downsampled audio. In his paper, Talkin explains that one should take each highlighted lag values of the downsampled audio and use parabolic interpolation to determine which lag values to consider in the original audio sample.

During the second pass I scan around each of these estimated peaks. I found that in some cases that the best correlation values were sometimes missed, so I decided to deviate somewhat from Talkin's proposed implementation. Rather than only scanning a few adjacent lag values around each proposed peak, I scanned a bigger portion of samples. This made my implementation more accurate, while still achieving good performance with the two-pass cross correlation method. As seen in Figure 14, the blue line represents the original implementation and the orange line shows my modified version. The dip in the middle of the pitch graph occurs because my second pass of the audio sample did not consider a lag value for that pitch, whereas for my later version of RAPT I scanned enough points to detect the ideal glottal pulse candidate.

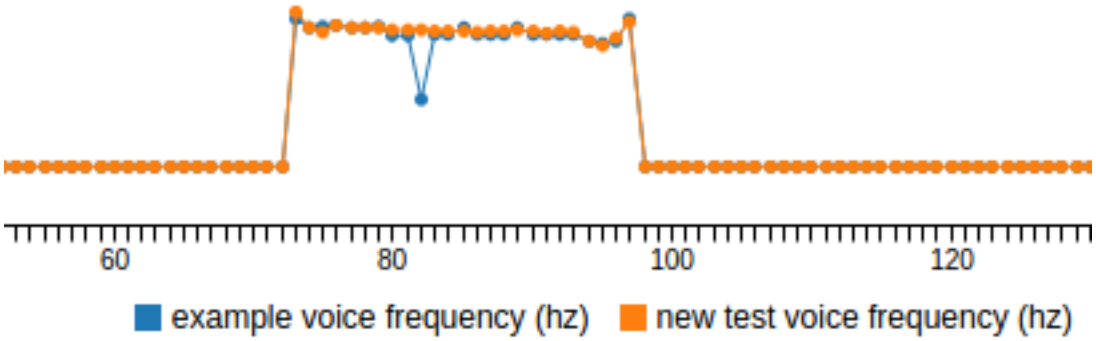

Figure 14. Comparison of original and modified RAPT implementation results.

Once I had my RAPT implementation identifying potential glottal pulses, I could begin work on the second part of the algorithm. I needed to ensure that repeating patterns of voiced speech would be considered as opposed to unvoiced speech or background noise.

4.3.2 Determining Voicing State With Dynamic Programming

The second part of RAPT involves determining the voicing state at each frame and selecting a specific correlated lag value to identify the vocal frequency. While there may be repeating patterns in each frame of the audio input, we only want to consider them when someone is speaking a voiced consonant. The pitch graph should also be a continuous function. That means that the pitch graph should appear as a continuous line and not have pitch values that jump from place to place.

Talkin specifies a dynamic programming approach to determining the most likely voicing state for each frame. At each step we must determine which candidate from the

NCCF output we should choose, or whether to select an unvoiced state. The dynamic programming also favors a candidate that maintains a continuous line for the pitch graph output.

The previous frame’s voicing state and glottal pulse candidates decide the state for the current frame, so the algorithm must first calculate the previous frame's potential states. This process continues until the very first sample. My implementation used a recursive function which gets the initial default costs passed in to start. The function proceeds to calculate the costs of selecting each potential candidate based on the previous frame's results.

```
def _get_next_cands(frame_idx, prev_candidates, nccf_results,
params, sample_rate):
   # best correlation for the current frame of the audio sample
   frame_max =
   _select_max_correlation_for_frame(nccf_results[frame_idx])
   final_candidates = []
   for candidate in nccf_results[frame_idx]:
      best_cost = None
      returned_path = None
      # at each step determine cost of selecting this candidate
      # based on its correlation value, relative to the best
      # correlation for this frame
      local_cost = _calculate_local_cost(candidate, frame_max,
                                         params, sample_rate)
       for prev_candidate in prev_candidates:
            # for each previous candidate, calculate the cost of
            # transitioning to this current candidate
            delta_cost = _get_delta_cost(candidate,
                                         prev_candidate[-1],
                                         frame_idx, params)
            total_cost = local_cost + delta_cost
            # pick best lowest transition cost. The returned_path
            # contains the history of selected candidates for each
            # frame
            if best_cost is None or total_cost <= best_cost:
                  best_cost = total_cost
                  returned_path = list(prev_candidate)
            returned_path.append((best_cost, candidate))
            final_candidates.append(returned_path)
      # determine the best choice for each candidate in this frame,
      # use the current frame's results to calculate the next frame
      next_idx = frame_idx + 1
      # recursive step <------
      if next_idx < len(nccf_results):
            return _get_next_cands(next_idx, final_candidates,
                                   nccf_results, params, sample_rate)
      return final_candidates
```

Figure 15. Python code that determines voicing state of a given frame.

Talkin's paper on RAPT specifies the rules for calculating the costs for voicing state transition and selecting a potential glottal pulse. This logic is contained within the _get_delta_cost method used in the example code above. My advisor, Paul Bamberg, suggested that I deviate from the formula in RAPT when calculating the transition cost between unvoiced and voiced states. Specifically, he suggested that I skip calculating the

Itakura distortion, which measures the similarity of two frequency spectra. (Soong & Sondhi, 2006). Talkin specifies it in his algorithm when determining the voicing state, but the algorithm's output seemed satisfactory without this calculation. In my case I did not need an extreme degree of accuracy. As long as the pitch graph I presented to users gave a reasonable representation of vocal pitch then I would not need additional complexity in my RAPT implementation. This also aided the performance of my algorithm since it did not need to make these additional calculations when finally determining pitch.

Once I implemented both parts of RAPT, I had a Python function that would return a pitch graph for a given audio input. The implementation closely resembled David Talkin's RAPT, with only a few alterations, and it performed well enough to be used for my web application. The Mandarin Tone Trainer could present a pitch graph after receiving audio input in a reasonable timeframe. Armed with a functioning pitch tracker, I could begin developing the web application itself.

4.4 Node.js Web Server and Backbone SPA

For the next part of the project, I needed a web application that could pass audio input to a running Python process. As stated earlier, I chose to run my application on a Node.js server. Using the Express library, I could set the desired port and routes where I expected HTTP requests to arrive from, and then determine what code to call in response.

The next step was to build a Single Page Application that would run within a client's browser. I used a preexisting library known as Backbone, which offers a simple framework for SPAs. The most notable feature of SPAs is that navigating to other pages involves dynamically loading new content without loading a new page in the browser.

Backbone offers a client-side web handler that changes the behavior of links. Rather than navigate the browser to a new page, it calls logic in Backbone's router object, as seen below:

```
// Set up global click event handler to use pushState for links
// use 'data-bypass' attribute on anchors to allow normal link
// behavior
$(document).on('click', 'a:not([data-bypass])', function(event) {
  var href = $(this).attr('href');
  var protocol = this.protocol + '//';
  if (href.slice(protocol.length) !== protocol) {
    event.preventDefault();
    router.navigate(href, true);
  }
});
```

Figure 16. SPA framework code that handles navigation within the web application.

The router is defined in the Backbone library and handles navigating to views internally within the client-side application. When a link is clicked, the address is checked against a set of routes to views within the Single Page Application. If the route matches, the router object will render the page with the new view without causing the browser to navigate away or reload the page. When an end user navigates to different pages of my application, the Backbone framework is simply rendering views within the same browser page and fetching data from the server asynchronously as needed.

4.5 User Interface

The interface of the Mandarin Tone Trainer was intentionally kept very simple, consisting of two pages. The first one presents some basic instructions to the user, as well as a few configuration options. One dropdown lets the user select a set of example words spoken in Mandarin by a specific person, allowing the user to choose a voice with a

matching pitch range. It also gives the user an opportunity to choose someone that speaks the words in a specific regional dialect.

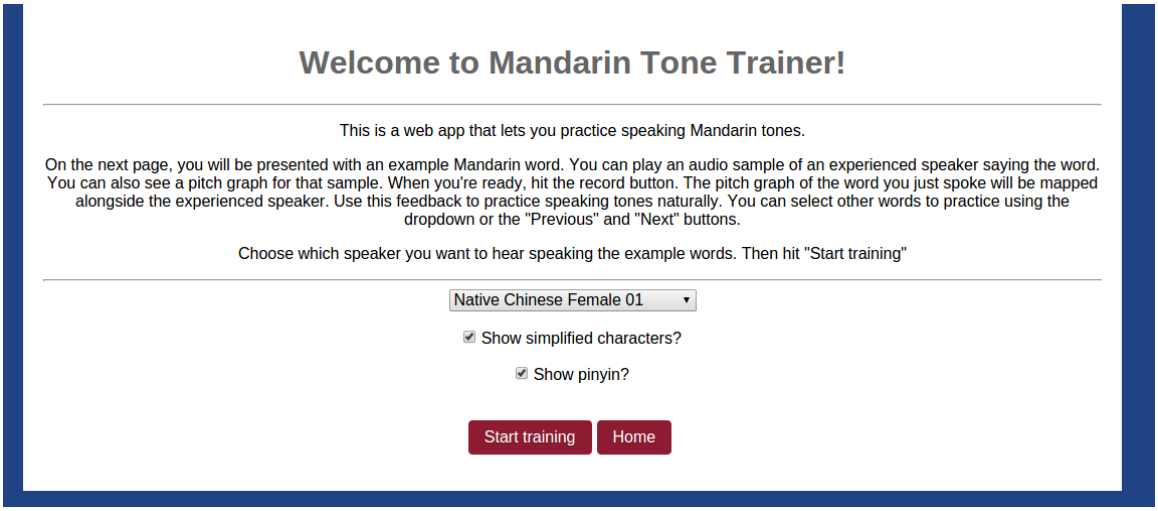

Figure 17. Initial page of the Mandarin Tone Trainer application.

For my initial implementation of the application I had a female native Mandarin speaker from China, and a male Taiwanese-American. This page also allows users to select the presentation of Mandarin text. The example words will be shown in either simplified or traditional Chinese characters.

Once a speaker profile has been selected, the user may click a button to continue onto the primary page of the application. It is here that I show users the pitch of a native speaker while speaking an example word in Mandarin. The word appears at the top of the screen, and the page provides audio playback as well as a visual representation of the speaker's pitch (Figure 18).

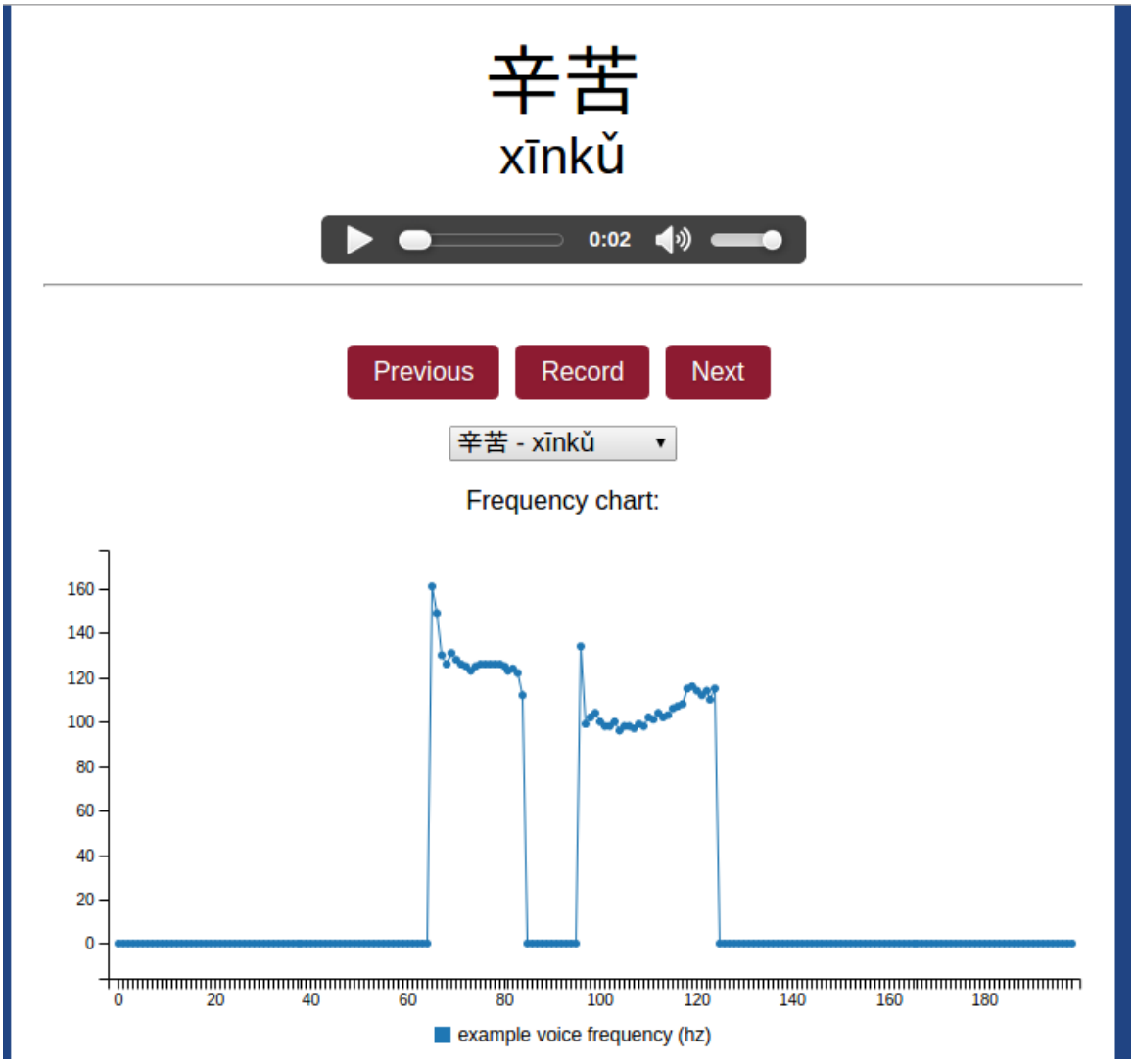

Figure 18. Main interactive page of the Mandarin Tone Trainer application.

The user then clicks a button to record their own voice. After a brief delay, the pitch graph of the user is charted alongside the pitch graph of the native speaker. The two pitch graphs are plotted in different colors so they can be visually compared. This visual feedback should help users as they practice speaking the example word.

Figure 19. The user's pitch graph overlaid onto the example speaker's pitch graph.

The user interface also provides buttons and a dropdown menu to select other example words. The user can select a specific word to practice or navigate through the entire collection. This allows users to navigate at their own pace through the set of example words.

The interface for this application has been kept deliberately simple. The page layout keeps the components of the page within one central column, which allows them

to display properly on a mobile device. The deliberate simplicity also means it could be integrated into other applications without much effort.

4.6 Storing and Presenting Spoken Examples to the User

The Mandarin Tone Trainer application relies on sets of example words. I needed to retrieve the audio and pitch graph for each example, as well as the relevant text for the word. Since this data was static in nature, I chose to store all of it in a simple database. I specifically chose to use SQLite for storing my data.

SQLite is a simple file-based relational database. Interactions are done using SQL syntax. Data is stored in tables and queried much as if it were stored on a database server, but the data itself is kept in a single file. Since the dataset for the project was relatively small and unlikely to change, a SQLite database was an ideal way to store the information. The data file can even be tracked in a source control system to be easily backed up or restored if there is a problem.

Once I had a system in place to store data, I could select which data would persist in my database file. I extracted the audio data for each spoken example, and stored the Mandarin and pinyin characters representing each word. Finally, I saved the output of my RAPT algorithm on each audio sample, which returned a set of points I could use to plot a pitch graph. It was more efficient for me to run my pitch tracker on each audio sample ahead of time and then store the output in the SQLite database.

```
sqlite> select ExampleId, SpeakerId, MandarinWord, PinyinWord, WavFile FROM Examples Limit 5;
ExampleId   SpeakerId   MandarinWord  PinyinWord  WavFile
----------  ----------  ------------  ----------  ----------
29          1           新            xīn         RIFFP`
30          1           酸            suān        RIFF
31          1           忙            máng        RIFF``
32          1           油            yóu         RIFF◆\
33          1           好            hǎo         RIFF◆e
sqlite>
```

Figure 20. Partial Schema of the Examples Table in SQLite.

Once I had stored each example provided by my volunteer Mandarin speakers, it was a simple process to present them to the user in the application.

The last step of this process was to determine how to display a pitch graph in the browser. I chose a popular data visualization library, D3, when presenting vocal frequency. D3 is a robust library used by many in the sciences to visualize data in the browser, with excellent support for plotting graphs. It also has many supporting libraries that focus on specific types of visualization. In my case I was able to quickly write a client-side handler that would plot a pitch graph, making it easy to add the user's pitch onto the same graph. By using a well-established visualization library, I was able to present information to the user simply and effectively. One additional benefit to this approach is that it displayed well on mobile devices.

4.7 Using WebRTC to Capture Audio Input

The main page of the Tone Trainer application needed to record audio from a user and present the speaker's pitch. To capture the audio input, I used the MediaStream JavaScript API as specified in the WebRTC standard. When a user first loads the application, it requests access to recording devices. Browsers that support WebRTC

present a dialog requesting permission to the user. Once the user has accepted, the application can record audio.

Figure 21. The browser prompting a user for access to a microphone.

The application does not begin recording until the user is ready to speak the example word. The "Record" button on the user interface triggers audio recording once clicked. The MediaStream API is used to collect audio information during this time period.

Once about two seconds have elapsed the application stops recording and begins to calculate the pitch information. My Python pitch tracker works on a WAV file as an input, so my application converts the audio to a WAV file and contacts the pitch tracker process running in Python.

I used a client-side library called RecorderJS to handle encoding the audio input into a WAV file. This library provided a fast and efficient way to encode the audio recorded using WebRTC. This WAV file data is generated while running in the client's browser, then the browser asynchronously sends the audio data to the web server. Next, the server contacts the Python process using ZeroRPC. The only thing the server needs to

do is invoke the Python method with the WAV audio as input. The RPC call reaches the running Python process which receives the request, runs RAPT on the audio, and returns a set of points that represents the user's pitch graph. The below method shows the simplicity of the process:

```
var sendFrequencyController = function(req, res) {
  //sendfreq - make RPC call w/ blob and get freq map for input
  var freqResults = {};
  var client = new zerorpc.Client();
  client.connect('tcp://127.0.0.1:4242');
  //req.files has reference to the audio file just uploaded -
need to open it, send data via RPC, and delete it once
  //we've finished.
  client.invoke('raptforfile', req.file.path, function(error,
rpcRes, more) {
    freqResults = rpcRes;
    res.json({'freqmap': freqResults});
    //at this point it should be safe to delete the file

    fs.unlinkSync(req.file.path);
  });
};
```

Figure 22. Server code that calls the Python process via ZeroRPC.

The web server receives this response from the Python process and then returns it to client, at which point the new pitch information is displayed to the user. Thanks to the efficiency of the RecorderJS library and the speed of my RAPT implementation, this process typically takes only a few seconds.

4.8 Obtaining Recorded Examples from Mandarin Speakers

Since I was already developing an application to record audio from users, I also decided to make a web application to record examples from Mandarin speakers remotely. I made a simple recording web application using the same components of the Mandarin Tone Trainer. I gave the address of the web application to anyone interested in recording samples. The application in the browser would present a series of example words to be

spoken, and record them in samples that lasted two seconds. I made sure that the length of the example recordings would match the timeframe that users had to speak each word. This meant that the pitch graph of the native speaker and a user would be of the same length. I did not plan to shift the data points for either pitch graph to make them match exactly.

By creating simple application alongside my main project, I was able to simplify obtaining samples from users and created a way to record more examples in the future. Although the recording equipment used by volunteers might not be uniform, the general recording process is comparable to how audio is captured in Mandarin Tone Trainer. Ideally the recorded examples should be close in quality and duration to the audio samples users generate while using my application.

Chapter 5 Summary and Conclusions

My goal to create a tool for practicing Mandarin tones online was a success. The Mandarin Tone Trainer provides visual feedback to users so that they learn to speak tones correctly. The user interface was written purely in JavaScript so that it could be easily integrated with MOOC courseware. Furthermore, the application responds in a reasonable timeframe and could be scaled to handle many more requests.

This project gave me an opportunity to work with new technologies and methods, including learning about signal processing and pitch tracking, and creating a Node.js web server that could communicate with Python processes. Although I have already identified areas that could use improvement, overall I feel that the Mandarin Tone Trainer would be a helpful tool for students learning to speak Mandarin.

5.1 Overview and Results

The primary objective of the Mandarin Tone Trainer was to create something that lets users practice speaking Mandarin tones. By presenting a graph mapping both the user and a native speaker's pitch, it does exactly that. A user can repeatedly practice speaking an example word with the native speaker's tone information as a guide, and the server responds quickly enough to be useful to a student learning over the web.

I definitely feel that the tool would be useful for online students. Having recorded examples from native speakers provides guidance, and the visual feedback from the pitch

graph shows how well a user is speaking the given word. This kind of practice is missing from most online Mandarin courses, and it would help make the learning experience more engaging.

5.1.1 Assessment of Pitch Tracker

I believe that the pitch tracker's output is satisfactory for the purpose of my project. For each recorded example spoken in Mandarin, the curve of the pitch graph matches the expected rise and fall of the tones in the word. The output appears to be correct, but it is susceptible to background noise or poor recording quality. One set of my recordings have a great deal of echo and background noise, which means there are obvious small errors in the graph. However, I have found that even in these situations the curve of the Mandarin tones is still comprehensible in the pitch graph. I have invited fellow Mandarin students to try out the application, and they report that they can easily make out the tones of the pitch graph. It would be better, however, to confirm my pitch tracker's output through more objective measures.

One way to verify the output of my pitch tracker is to compare it to the results from a different program. I had already discovered the Praat application while preparing for this project, so I decided to compare my pitch tracker results with the output from Praat.

While recording one of my volunteer Mandarin speakers, I had her record an additional set of words I did not intend to present to users of the Tone Trainer application. These recordings represent a validation set of data I planned to use for my

final assessment of my pitch tracker. The following figures show the resulting pitch graphs for these words, and the comparable output in Praat.

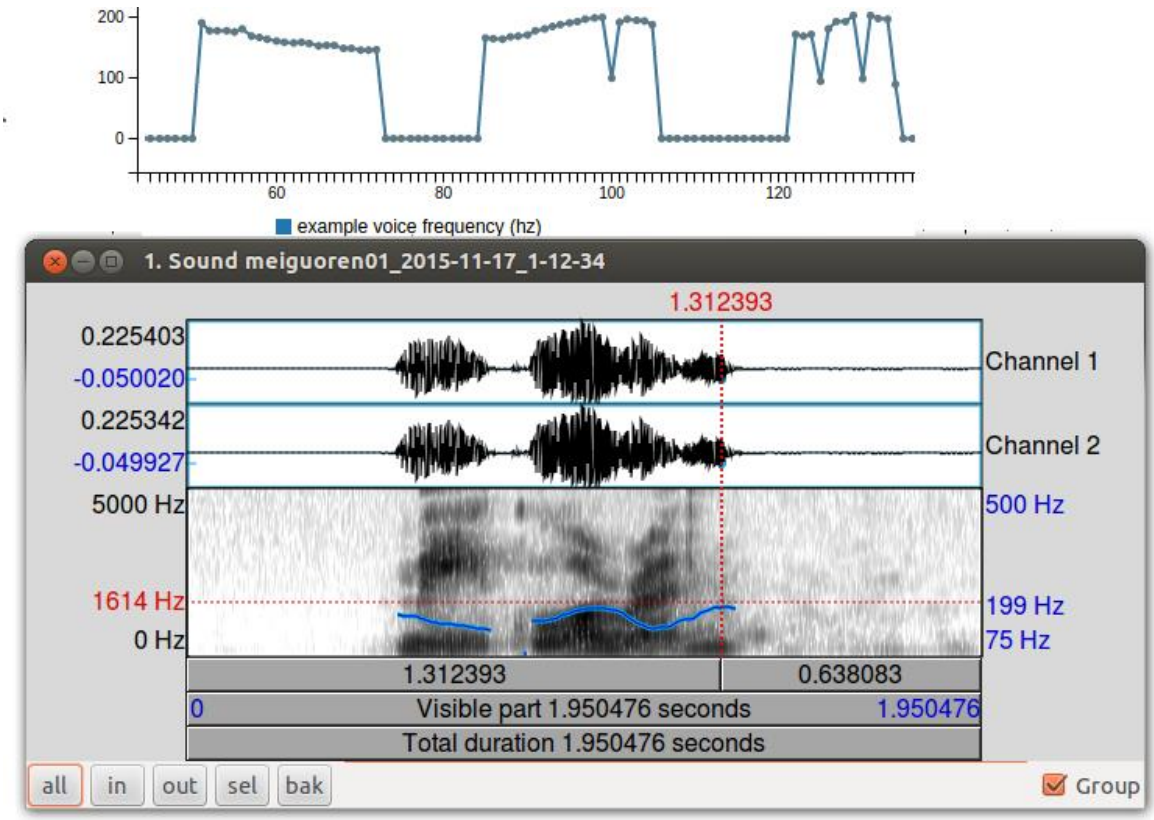

Figure 23. Comparison of Mandarin Tone Trainer's pitch graph above with output from Praat below.

Overall, while there were some errors and susceptibility to noise in my implementation, the pitch information from my version of RAPT correlated well with what was I saw in a different pitch tracker. The main difference between the two is that RAPT explicitly treats unvoiced segments as having zero frequency, whereas Praat is more likely to treat the intermediate unvoiced portions of a word as a single utterance. This is most obvious in Figure 23 where the “r” of the word “měiguórén” is treated as having pitch in Praat but not in my RAPT implementation. Ultimately, the frequency

values and rate of change in each tracker's output, as seen in Figures Figure 23 and Figure 24, are visually similar. Since the pitch graph in Mandarin Tone Trainer is used for visual guidance of students, I think this meets the criteria for success.

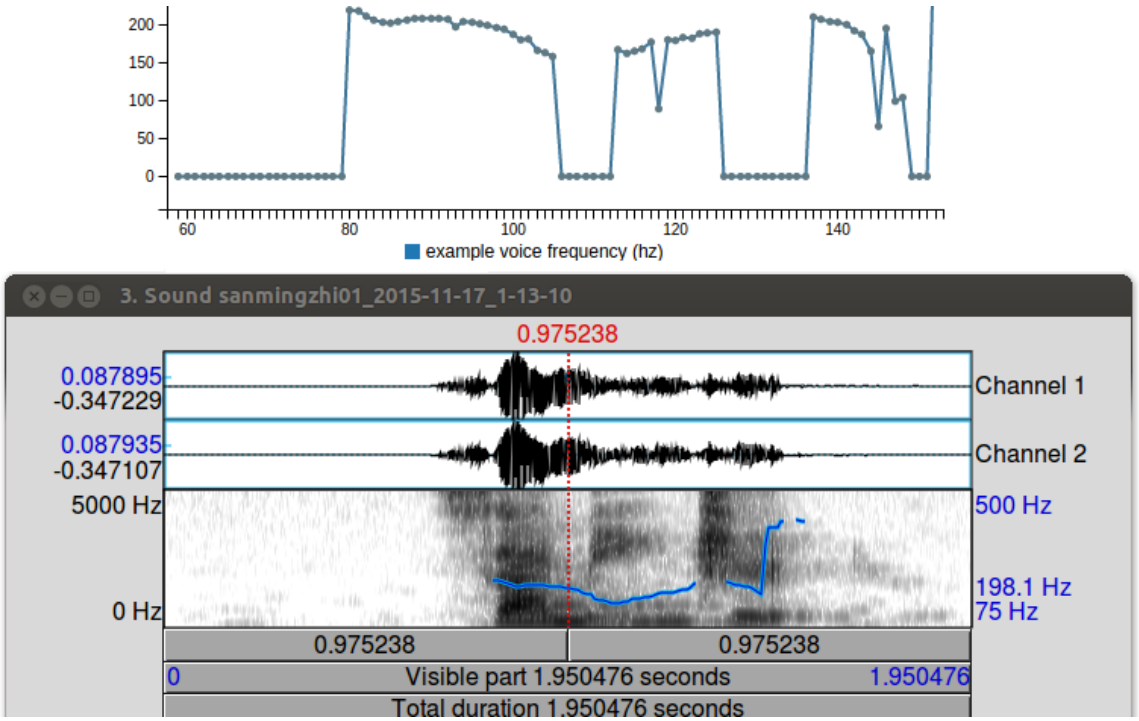

Figure 24. Additional comparison of Mandarin Tone Trainer and Praat.

5.1.2 User Feedback

Because I was taking an introductory course in Mandarin, I had a group of students to confer with to get feedback on my project. I received a good deal of positive feedback from my fellow students. They appreciated the concept of a tool for practicing Mandarin tones. The pitch information was decipherable to them, and based on our informal discussions they seemed to like the application. They would have liked the

application to inform them if their pitch did not match that of a native speaker, but overall they found the Mandarin Tone Trainer to be a useful learning tool.

I also showed the application to coworkers at my office. While they were not well-versed in the peculiarities of Mandarin tones, they found the presentation of the pitch graph to be intuitive. They could see that the graph represented their vocal pitch plotted against that of the example audio. They found it interesting to have such a tool available within a web browser and even on a mobile phone.

Overall the feedback I received on my project was positive. My fellow students were happy to see a tool for practicing Mandarin tones. Being able to practice tones outside of the classroom is desirable to both online students and those that are taking classes in person.

5.2 Challenges

5.2.1 WebRTC Browser Support and Changing Standards

One obstacle while working on the Tone Trainer was the lack of full integration with all browsers. Although WebRTC is now supported by most major browsers, it is still not supported by the Safari browser developed by Apple. This means that Apple devices running iOS cannot use WebRTC to record audio for my application, and users with desktop computers made by Apple must use browsers other than Safari in order for recording to function correctly. There is a large portion of mobile users that cannot fully use my application at this time, and others must download new browsers in order for Mandarin Tone Trainer to function.

Although most browsers do support WebRTC, the standard is still in flux. Due to security concerns, the Google Chrome development team has changed how their browser handles WebRTC functionality. Chrome will only allow web sites provided via HTTPS to use WebRTC functionality. A page loaded with HTTP will not be allowed to request access to a user's recording devices. This change happened half way through my project, forcing me to make changes to my server to accommodate this. WebRTC is still a new standard and may yet change again. Developing a web application that relies on a new standard like WebRTC is difficult, and changes in the future may break my application.

5.2.2 Implementing RAPT

I found that it was not difficult to create a roughly functioning implementation of David Talkin's algorithm, but it was difficult to find and identify bugs. The initial pitch graphs returned by my Python process running RAPT had the pitch contours I was expecting to see, but many inaccuracies in the pitch graph which appeared to be a result of noise were actually the result of defects in my program. It was a difficult task to track down errors, since the correlation data and voicing state costs calculated during RAPT are quite large for a high quality WAV audio sample. Once I realized I needed to scrutinize the intermediate data generated by the algorithm, I found that visualization tools helped pinpoint potential problems.

I discovered a JavaScript visualization library called Plotly, which had a simple interface for creating a heatmap. This allows one to plot values on a 2-dimensional graph with each point having a different color depending on the severity of a 3rd value at each data point. This is ideal for presenting the output of the NCCF in RAPT. For each lag at each frame there is a correlation value. The closer the correlation is to the value 1.0, the

more likely there is a repeating waveform at that frequency. Figure 12 in Part 4.3 of this thesis shows one such example of a NCCF heatmap. Using this kind of visualization tool makes it much easier to assess which frequencies are being considered for voiced speech by RAPT.

I also used a visualization library called SpectrogramJS. A spectrogram shows the entire spectrum of frequencies present in an audio sample, with those of higher amplitude appearing more brightly. My advisor showed me that with the right expertise one can identify the words being spoken based on a spectrogram. It also could help identify the proper pitch of each word. Properly identifying words and pitch from a spectrogram requires some expertise, however. I didn't get a chance to learn how to read them properly during my project, but given more time I could use spectrograms to better validate my pitch tracker's output.

Although my application is functional, I would like to see even better accuracy. In a good recording environment my pitch tracker works well, but background noise and poor recording tools generate less than ideal results. More work could be done to improve my implementation of RAPT.

5.2.3 Collecting Samples of Spoken Mandarin Words

Obtaining example audio from native Mandarin speakers was difficult, even with the application I created to help the process. One of my primary sets of examples has a great deal of echo in the background, which affects the quality of the resulting pitch graphs. It was also difficult to gather samples from multiple sources, and some volunteers did not have a microphone beyond the one in their mobile phones. It did not help that my

recording application did not work on iPhones, since Safari does not support WebRTC. As a result, I only used two sets of spoken examples for my project.

5.3 Suggested Improvements

I feel there are two areas to improve the Tone Trainer. The first of these is to provide a better set of examples to users. I focused on a basic set of words with at least one combination of each type of tone. I did not select any examples that were three characters or longer. This means that my set of example words did not involve some of the more complex combinations of tones. Mandarin has special rules for certain tone combinations, known as sandhi, that change the standard rules for speaking certain words. Although some of the two character words provided some basic use cases for these tone sandhi, longer words would have introduced other more complicated examples.

A better set of example words would include both basic use cases as well as more advanced examples. Longer words or even phrases might give a student a more organic way to practice speaking tones. Saying a word by itself is a different experience than speaking an entire sentence. I would prefer to add even more examples for my application to make it a better learning tool.

At the time of this project my Tone Trainer application offers examples spoken by people from Taiwan and China. I realized while working on this project that there are regional differences to how Mandarin is spoken. Some tone sandhi rules are different between Taiwanese and mainland Chinese speakers of Mandarin (Zhang, Lai, & Sailor, 2011). I am glad that I have two speakers that come from differing regions, but I would

like more speakers from other regions as well. This would give the Mandarin Tone Trainer a better range of speakers for users to compare themselves with. I would also like to learn more about regional differences in tones spoken within China itself. My initial selection of speakers was a good start, but I would prefer to have more variety in regional dialects.

The other improvement that would make the Tone Trainer useful to online courses would be some way to assess how well a student's pitch graph correlates with the example speaker's graph. This was suggested by some of my fellow students while I demonstrated my application. While it is easy to visually compare one's pitch graph with a native speaker's in Tone Trainer, it would be even better to get an automated judgment of how well the two correlate.

One possible way to do this would be to use a correlation function similar to how the NCCF is used in RAPT to identify repeating waveforms. We could run a correlation function on the native speaker's graph and how well it correlates with the user's pitch graph. One thing to consider is that the user may not speak in the same pitch range as the native speaker. Any comparison would have to account for the natural vocal differences between the user and the native speaker. The key challenge with comparing two pitch graphs is measuring the dissimilarity between the two and defining the threshold for whether a user's input is correct or not. Much research has been done on this topic and there are efficient algorithms I could use, such as Michael Morse and Jignesh Patel's Fast Time Series Evaluation method. It uses a scoring model that could be tuned to measure the difference between graphs while accounting for differences in vocal pitch range and any noise in the user's audio input (Morse, 2007).

Once I had a way to compare the user's pitch graph with the native speaker's example, the interface could report whether the word was spoken correctly or not. I feel like this feature would help to integrate it with online classrooms. For example, a MOOC lesson could require a certain amount of correctly pronounced tones before proceeding. MOOCs already have quizzes that require a certain level of success before advancing, and tone practice would be a nice complement to that feature. I was unable to do this for my project, but this seems like the next step towards making a useful learning tool.

5.4 Conclusion

I think the web has a great deal of potential in helping students learn new languages on their own. Interacting with people and getting direct feedback from them is still the best way to learn, but there will be times where people cannot get direct access to speakers of a new language. Students need tools to learn and practice languages when they can't interact with other people directly, and I think the web can offer these.

Learning a tonal language like Mandarin is very difficult when someone has grown up speaking a non-tonal language like English. As someone learning Mandarin as a new language, I need to build up tonal instincts in order to speak the language naturally. Having a tool to practice speaking words while still getting meaningful feedback is very helpful.

I was able to create a useful application for this project on my own. The web is quickly becoming a platform where it is easy to develop and distribute learning tools, which makes me feel that it is a promising frontier for education in general. It has the potential to reach students all over the world and help them practice speaking new

languages. The Mandarin Tone Trainer is just one potentially valuable example of what is possible with today's web technology.

Chapter 6 Glossary

D3.js - A popular Javascript library used for dynamic data visualizations.

Fundamental frequency (f_0) - This is a property of a periodic repeating waveform. The rate of the largest pattern in the periodic waveform is the fundamental frequency. This correlates well with how humans perceive pitch.

Glottal pulse - The single cycle of vibration of the vocal chords, which causes a periodic vibration of the air that produces sound during voiced speech.

MediaStream - The MediaStream Processing API is a W3C standard that is only one component of the WebRTC standard. It is concerned with providing audio and video streams to client web applications.

MOOC - A Massively Open Online Course, or MOOC, is a term applied to large online courses with unrestricted enrollment policies.

NCCF – A normalized cross-correlation function, or NCCF, is a function that compares a portion of a waveform with another portion of a signal, and returns the level of similarity between the two.

Node - A shortened name for Node.js, a server-side Javascript run-time environment.

Pinyin - A phonetic alphabet written in Latin characters used to convey the pronunciation and tone of Chinese characters.

Simplified Chinese Characters - This is a standardized alphabet introduced in the People's Republic of China in the 1950's. It is used in China whereas other Mandarin speaking countries such as Taiwan use the "traditional" alphabet.

RPC - A remote procedure call, which is a technique for one system to invoke a program on another system via a simplified interface.

SQLite - A simple relational database management system that stores data in a local file rather than on a separate server.

SPA - A Single Page Application, or SPA, is a web application where the client's browser runs a client-side framework that dynamically navigates between pages without refreshing the browser when loading new pages.

Tone sandhi - These are rules for alternative tonal pronunciations of certain characters depending on other tones that are present in a word. For example if two 3rd-tone characters are present in a word, the second one is pronounced as a rising 2nd-tone.

W3C - The World Wide Web Consortium, or W3C, is an organization that manages communication and technology standards used on the web.

WebRTC - Web Real Time Communication, or WebRTC, is a media and communication API proposed by the W3C and developed by Google. It supports obtaining video and microphone input within a browser as well as streaming this data to other browsers to support video and audio conferencing.

Chapter 7 References

Bárcena E., Read T., Martín-Monje E., & Castrillo M.D. (2014). Analysing student participation in Foreign Language MOOCs: a case study, in proceedings of the European MOOC Stakeholder Summit, Lausanne, 10-12 February 2014, pp. 11-12. Retrieved from http://www.emoocs2014.eu/sites/default/files/Proceedings-Moocs-Summit-2014.pdf

Bergkvist, A., Burnett, D. C., Jennings, C., & Narayanan, A. (2012). WebRTC 1.0: Real-time communication between browsers. *Working draft, W3C*, *91*.

Chang, R., Hung, Y., & Lin, C. (2015). Survey of learning experiences and influence of learning style preferences on user intentions regarding MOOC s. *British Journal of Educational Technology, 46*(3), 528-541.

Chun, D. (1989). Teaching tone and intonation with microcomputers. *CALICO Journal*, *7*(1), 21-46.

Chun, D. M., Jiang, Y., & Ávila, N. (2013). Visualization of tone for learning Mandarin Chinese. In *Proceedings of the 4th Pronunciation in Second Language Learning and Teaching Conference, Ames, IA: Iowa State University*.

Gerhard, D. (2003). *Pitch extraction and fundamental frequency: History and current techniques* (pp. 2003-06). Regina: Department of Computer Science, University of Regina.

Gouaillard, A. (2015, August 29). Webrtc in Safari? Retrieved December 21, 2015, from http://webrtcbydralex.com/index.php/2015/08/29/webrtc-in-safari/

Loreto, S., & Romano, S. (2012). Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts. *Internet Computing, IEEE, 16*(5), 68-73.

Morse, M. D., & Patel, J. M. (2007). An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 569-580). ACM.

Soong, F., & Sondhi, M. (2006). A frequency-weighted itakura spectral distortion measure and its application to speech recognition in noise. *STAR, 44*(2), STAR, 27 Jan. 2006, Vol.44(2).

Talkin, D. (1995). A robust algorithm for pitch tracking (RAPT). *Speech coding and synthesis*, *495*, 518.

Van Der Walt, S., Colbert, S., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering, 13*(2), 22-30.

Ventura, P., Bárcena, E., & Martín-Monje, E. (2014). Analysis of the Impact of Social Feedback on Written Production and Student Engagement in Language Moocs. *Procedia - Social and Behavioral Sciences, 141*, 512-517.

Zhang, J., Lai, Y., & Sailor, C. (2011). Modeling Taiwanese Speakers' Knowledge of Tone Sandhi in Reduplication. *Lingua: International Review of General Linguistics, 121*(2), 181-206.

Appendix 1 Application Code

The code for Mandarin Tone Trainer consists of two distinct sections. There is a Python project called “pyrapt” that handles the logic for tracking pitch. There is also the primary web application, “tonetrainer”, which runs on a Node.js server. Both of these separate projects are hosted on GitHub, which provides free source control management. The code for the two sections are contained within the two repositories listed below. GitHub’s web interface lets users view code files and navigate the directory structure of each project. I have also provided a basic description of each project’s structure to aid in understanding each codebase.

- tonetrainer: https://github.com/dgaspari/tonetrainer
- pyrapt: https://github.com/dgaspari/pyrapt

Tonetrainer

This is a web server that serves a single page application. The server startup is invoked by the server.js file at the root of the repository. Logic that handles the behavior of the web server itself can be found within the “server/” directory, while the logic pertaining to views and scripts that load on the client browser can be found in the “client/” directory.

Pyrapt

This is a Python module. The primary code for the module resides in the “pyrapt/” directory of the code repository. There are two helper elements, nccfparams and raptparams, that define constants used in the RAPT algorithm. The primary code of the module exists in “pyrapt.py”. The process that listens for requests from the tonetrainer server can be found in the “tonetrainer.py” file within the “server” directory. It is that code which is running on a Python process on my application server. It handles invoking the RAPT module and responding to the server via ZeroRPC.